Efficient Reinforcement Learning in Probabilistic Reward Machines

Xiaofeng Lin   Xuezhou Zhang

Boston University

AAAI 2025

Non-Markovian Rewards

Markovian Assumption in RL

- Rewards depend **only** on the current state and action.

Non-Markovian Rewards

Markovian Assumption in RL

- Rewards depend **only** on the current state and action.

Reality: Many Tasks Require Historical Context

- Rewards depend on the **sequence** of actions/states.

Non-Markovian Rewards

Markovian Assumption in RL

- Rewards depend **only** on the current state and action.

Reality: Many Tasks Require Historical Context

- Rewards depend on the **sequence** of actions/states.

For example...

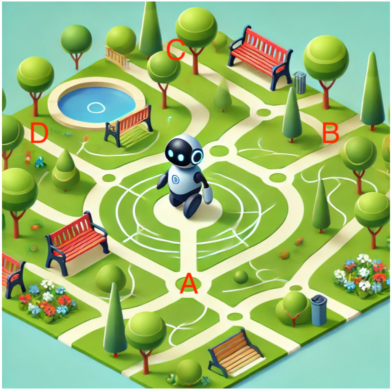
Non-Markovian Rewards

Task: make a coffee and deliver to office

Non-Markovianity: robot is only rewarded after making a coffee and delivering it.

Non-Markovian Rewards

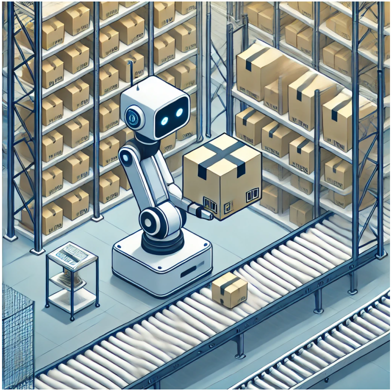
Task: patrolling park in order



Non-Markovianity: robot is only rewarded after patrolling location A, B, C and D in order.

Non-Markovian Rewards

Task: pick up item and deliver it



Non-Markovianity: robot is only rewarded after picking up item and delivering it.

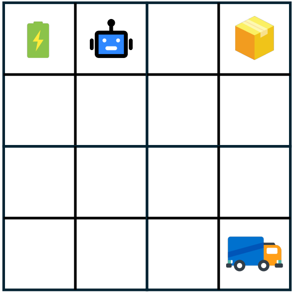
Deterministic Reward Machine

Problem: how do we reward such Non-Markovian behaviors?

Deterministic Reward Machine

Problem: how do we reward such Non-Markovian behaviors?

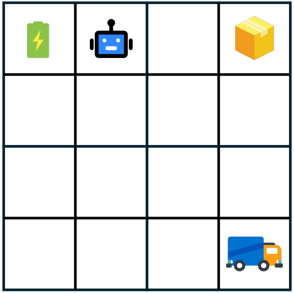
Task: collect and deliver it to



Deterministic Reward Machine

Problem: how do we reward such Non-Markovian behaviors?

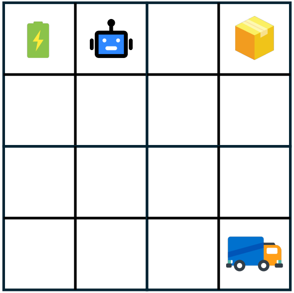
Task: collect  and deliver it to 

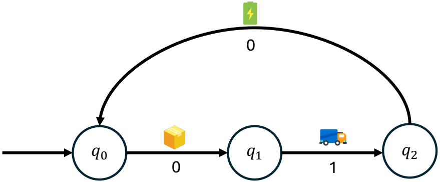


Deterministic Reward Machine

Problem: how do we reward such Non-Markovian behaviors?

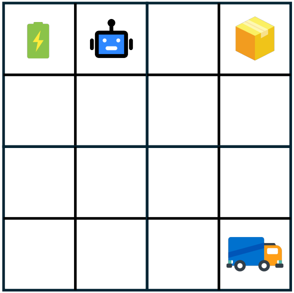
Task: collect  and deliver it to 

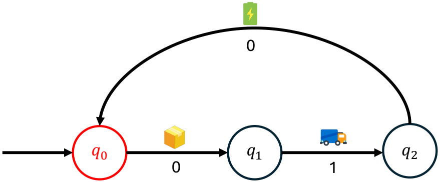




DRM in Action

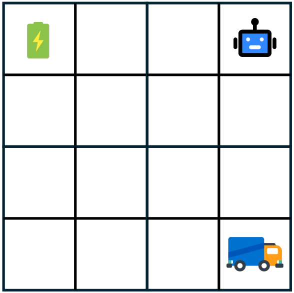
Task: collect and deliver it to

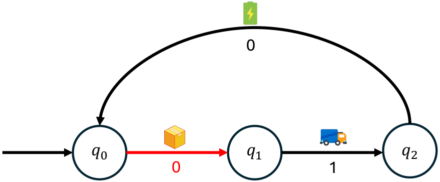




DRM in Action

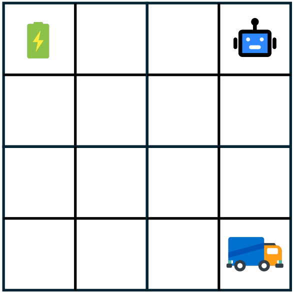
Task: collect  and deliver it to 

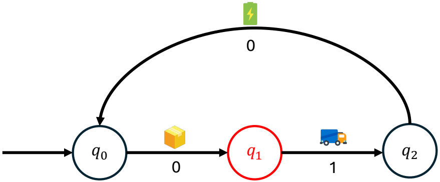




DRM in Action

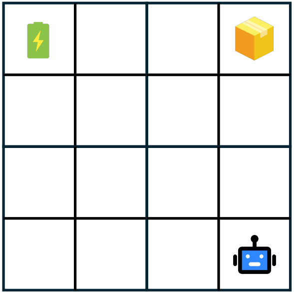
Task: collect and deliver it to

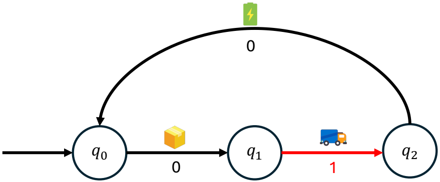




DRM in Action

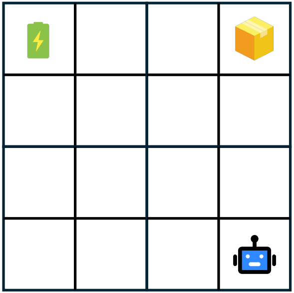
Task: collect and deliver it to

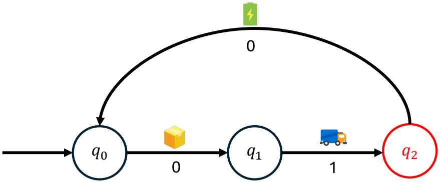




DRM in Action

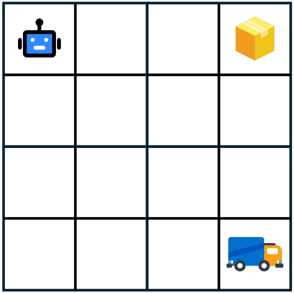
Task: collect and deliver it to

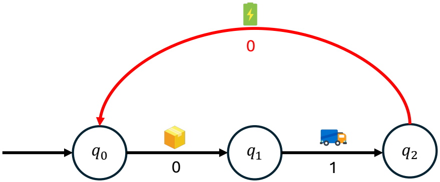




DRM in Action

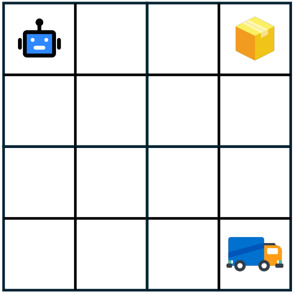
Task: collect and deliver it to

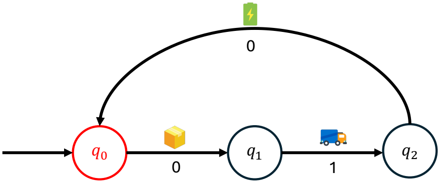




DRM in Action

Task: collect and deliver it to

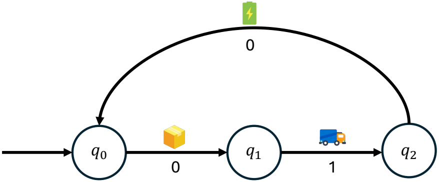




Deterministic Reward Machine

Deterministic Reward Machine (DRM)

- **finite set of states** $\mathcal{Q}$
- **atomic propositions**: $\mathcal{P}$
  e.g. $\mathcal{P} = \{$ , ,  $\}$
- **a deterministic transition function:**
  $\tau : \mathcal{Q} \times 2^{\mathcal{P}} \to \mathcal{Q}$,
  e.g. $\tau(q_0,$ $) = q_1$
- **A reward function**:
  $\nu : \mathcal{Q} \times 2^{\mathcal{P}} \to \Delta_{[0,1]}$
  e.g. $\nu(q_1,$ $) = 1$



Icarte, Rodrigo Toro, et al. "Using reward machines for high-level task specification and decomposition in reinforcement learning." International Conference on Machine Learning. PMLR, 2018.

Probabilistic Reward Machine

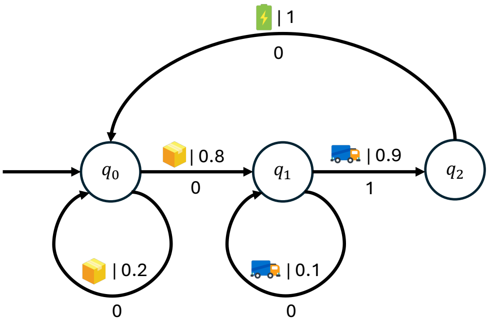
Problem:  is not always available,  is not always ready.

Probabilistic Reward Machine

Problem: 📦 is not always available, 🚚 is not always ready. 🤖?

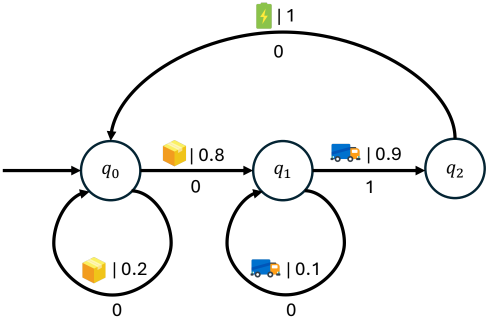
Probabilistic Reward Machine

Problem: 📦 is not always available, 🚚 is not always ready. 🤖?



Probabilistic Reward Machine

Problem: 📦 is not always available, 🚚 is not always ready. 🤖?



Previous experience: 80% 📦 is available, 90% 🚚 is ready.

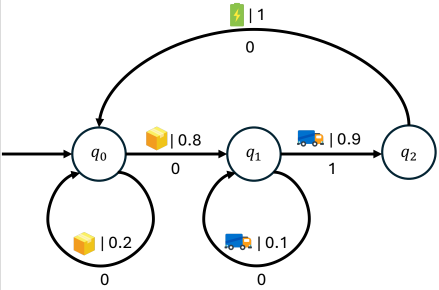
Probabilistic Reward Machine (PRM)

PRM

- **finite set of states** $\mathcal{Q}$
- **atomic propositions**: $\mathcal{P}$
  e.g. $\mathcal{P} = \{$🔋, 📦, 🚚$\}$
- **a** **probabilistic** **transition function:** $\tau : \mathcal{Q} \times 2^{\mathcal{P}} \to \Delta_{\mathcal{Q}}$,
  e.g.

$$\tau(q_0, \text{📦}) = \begin{cases} q_1, & w.p. 0.8 \\ q_0, & w.p. 0.2 \end{cases}$$

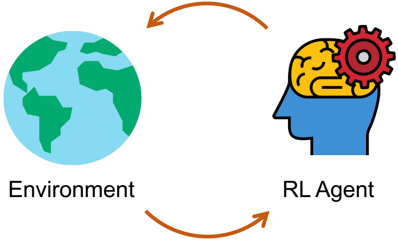
- **A reward function**:
  $\nu : \mathcal{Q} \times 2^{\mathcal{P}} \times \mathcal{Q} \to \Delta_{[0,1]}$
  e.g. $\nu(q_1, \text{🚚}, q_2) = 1$



Dohmen, Taylor, et al. "Inferring probabilistic reward machines from non-markovian reward signals for reinforcement learning." Proceedings of the International Conference on Automated Planning and Scheduling. Vol. 32. 2022.

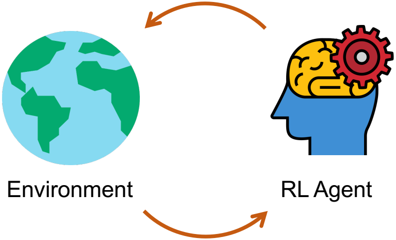
Markov Decision Process (MDP) with PRM

For timestep $h = 1, \dots, H$
Observation: $o_h \in \mathcal{O}$, State of PRM: $q_h \in \mathcal{Q}$
Action: $a_h \in \mathcal{A}$



Markov Decision Process (MDP) with PRM

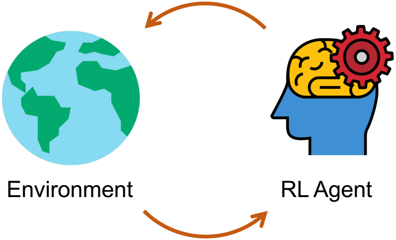
For timestep $h = 1, \ldots, H$
Observation: $o_h \in \mathcal{O}$, State of PRM: $q_h \in \mathcal{Q}$
Action: $a_h \in \mathcal{A}$



Next Observation: $o_{h+1} \sim p(\cdot | o_h, a_h)$

Markov Decision Process (MDP) with PRM

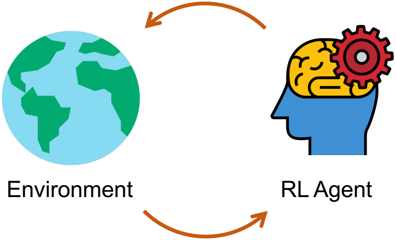
For timestep $h = 1, \ldots, H$
Observation: $o_h \in \mathcal{O}$, State of PRM: $q_h \in \mathcal{Q}$
Action: $a_h \in \mathcal{A}$



Next Observation: $o_{h+1} \sim p(\cdot|o_h, a_h)$
Event: $\sigma_h = L(o_h, a_h, o_{h+1}) \in 2^{\mathcal{P}}$

Markov Decision Process (MDP) with PRM

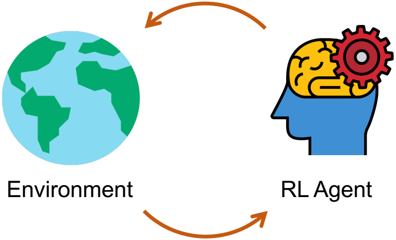
For timestep $h = 1, \ldots, H$
Observation: $o_h \in \mathcal{O}$, State of PRM: $q_h \in \mathcal{Q}$
Action: $a_h \in \mathcal{A}$



Next Observation: $o_{h+1} \sim p(\cdot | o_h, a_h)$
Event: $\sigma_h = L(o_h, a_h, o_{h+1})$
Next State of PRM: $q_{h+1} \sim \tau(\cdot | q_h, \sigma_h)$

Markov Decision Process (MDP) with PRM

For timestep $h = 1, \dots, H$
Observation: $o_h \in \mathcal{O}$, State of PRM: $q_h \in \mathcal{Q}$
Action: $a_h \in \mathcal{A}$

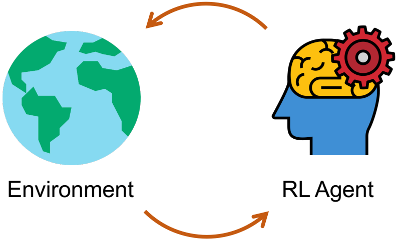


Next Observation: $o_{h+1} \sim p(\cdot|o_h, a_h)$
Event: $\sigma_h = L(o_h, a_h, o_{h+1})$
Next State of PRM: $q_{h+1} \sim \tau(\cdot|q_h, \sigma_h)$
Reward: $r_h = \nu(q_h, \sigma_h, q_{h+1})$

Xu, Zhe, et al. "Joint inference of reward machines and policies for reinforcement learning." Proceedings of the International Conference on Automated Planning and Scheduling. Vol. 30. 2020.

Markov Decision Process (MDP) with PRM

For timestep $h = 1, \ldots, H$
Observation: $o_h \in \mathcal{O}$, State of PRM: $q_h \in \mathcal{Q}$
Action: $a_h \in \mathcal{A}$

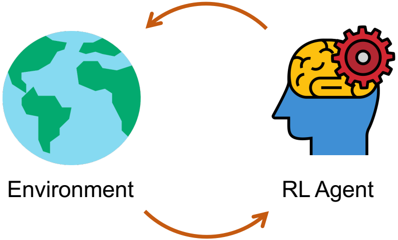


Next Observation: $o_{h+1} \sim p(\cdot | o_h, a_h)$
Event: $\sigma_h = L(o_h, a_h, o_{h+1})$
Next State of PRM: $q_{h+1} \sim \tau(\cdot | q_h, \sigma_h)$
Reward: $r_h = \nu(q_h, \sigma_h, q_{h+1})$

!
We know PRM for lots of tasks.

Markov Decision Process (MDP) with PRM

For timestep $h = 1, \ldots, H$
Observation: $o_h \in \mathcal{O}$, State of PRM: $q_h \in \mathcal{Q}$
Action: $a_h \in \mathcal{A}$

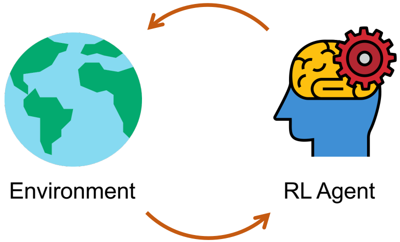


Next Observation: $o_{h+1} \sim p(\cdot | o_h, a_h)$
Event: $\sigma_h = L(o_h, a_h, o_{h+1})$
Next State of PRM: $q_{h+1} \sim \tau(\cdot | q_h, \sigma_h)$
Reward: $r_h = \nu(q_h, \sigma_h, q_{h+1})$

🤖!
We know PRM for lots of tasks.
80% 📦 is available, 90% 🚚 is ready.

Markov Decision Process (MDP) with PRM

For timestep $h = 1, \dots, H$
Observation: $o_h \in \mathcal{O}$,
State of PRM: $q_h \in \mathcal{Q}$
Action: $a_h \in \mathcal{A}$



Next Observation: $o_{h+1} \sim p(\cdot|o_h, a_h)$
Event: $\sigma_h = L(o_h, a_h, o_{h+1})$
Next State of PRM: $q_{h+1} \sim \tau(\cdot|q_h, \sigma_h)$
Reward: $r_h = \nu(q_h, \sigma_h, q_{h+1})$

How do we learn when we know PRM?

Expected total reward:

$V(\pi) = \mathbb{E}_{P,\pi}[r_1 + \cdots + r_H]$

Optimal Policy:

$\pi^\star = \arg\max_\pi V(\pi)$

Regret:

How well the policy perform against the optimal policy in K episodes?
$\text{Regret}(K) := \sum_{k=1}^{K} (V(\pi^*) - V(\pi_k))$

Cross-Product MDP

Let $\mathcal{S} = \mathcal{Q} \times \mathcal{O}$, and for $s = (q, o)$, $s' = (q', o') \in \mathcal{S}$ and $a \in \mathcal{A}$:

$$P(s' \mid s, a) = p(o' \mid o, a)\, \tau(q' \mid q, L(o, a, o'))$$

$$R(s, a) = \sum_{o' \in \mathcal{O}, q' \in \mathcal{Q}} p(o' \mid o, a)\, \nu(q, L(o, a, o'), q').$$

Cross-Product MDP

Let $\mathcal{S} = \mathcal{Q} \times \mathcal{O}$, and for $s = (q, o)$, $s' = (q', o') \in \mathcal{S}$ and $a \in \mathcal{A}$:

$$P(s' \mid s, a) = p(o' \mid o, a)\, \tau(q' \mid q, L(o, a, o'))$$

$$R(s, a) = \sum_{o' \in \mathcal{O}, q' \in \mathcal{Q}} p(o' \mid o, a)\, \nu(q, L(o, a, o'), q').$$

We can get

$$\mathcal{M}_{cp} = (\mathcal{S}, \mathcal{A}, P, R)$$

! Now the rewards and transition are all Markovian w.r.t. s

Cross-Product MDP

We can apply off-the-shelf RL algorithm to $\mathcal{M}_{cp}$

Cross-Product MDP

We can apply off-the-shelf RL algorithm to $\mathcal{M}_{cp}$

How do these algorithms perform?

Cross-Product MDP

We can apply off-the-shelf RL algorithm to $\mathcal{M}_{cp}$

How do these algorithms perform?

Regret grows not slower than $\Omega(\sqrt{QOAH^2K})$!

Auer, Peter, Thomas Jaksch, and Ronald Ortner. "Near-optimal regret bounds for reinforcement learning." Advances in neural information processing systems 21 (2008).
Bourel, Hippolyte, et al. "Exploration in reward machines with low regret." International Conference on Artificial Intelligence and Statistics. PMLR, 2023.

Cross-Product MDP

We can apply off-the-shelf RL algorithm to $\mathcal{M}_{cp}$

How do these algorithms perform?

Regret grows not slower than $\Omega(\sqrt{QOAH^2K})$!

The established regret lower bound is $\Omega(\sqrt{H^2OAK})$ for MDPs with DRMs, $\sqrt{Q}$ slower.

Auer, Peter, Thomas Jaksch, and Ronald Ortner. "Near-optimal regret bounds for reinforcement learning." Advances in neural information processing systems 21 (2008).
Bourel, Hippolyte, et al. "Exploration in reward machines with low regret." International Conference on Artificial Intelligence and Statistics. PMLR, 2023.

Algorithm Template

- For episodes $k = 1, \ldots, K$

1. Use data buffer D to estimate transition functions $\hat{p}_k$.
2. **[Model Estimation]** Construct $\hat{P}_k$ and $\hat{R}_k$ given $\hat{p}_k$ and the knowledge of PRM. Construct bonus reward b_k.
3. **[Planning]** Find the optimal policy π_k of MDP $(\hat{P}_k, \hat{R}_k + b_k)$.
4. agent plays policy π_k, collects data and appends it to D

Bonus Design

Denote $W_h : \mathcal{Q} \times \mathcal{O} \times \mathcal{A} \times \mathcal{O} \to \mathbb{R}$ a function that measures the expected return when being in state (q, o), executing action a at time step $h-1$ and observing o' at time step h. W is defined as follows:

$$W_h(q, o, a, o') = \sum_{q' \in \mathcal{Q}} \tau(q'|q, L(o, a, o'))V_h(q', o')$$

The estimation error $(\widehat{P}_k^{\pi_k} - P_h^{\pi_k})V^*_{h+1}$ can be translated to the estimation error in the observation space $(\widehat{p}_k^{\pi_k} - p^{\pi_k})W^*_{h+1}$.

- Bonus design: Bernstein-style bonus reward using W_k to ensure V_k is upper bound of V^*. The regret grows in the order of $|O|$ instead of $|Q||O|$

Theoretical Guarantee

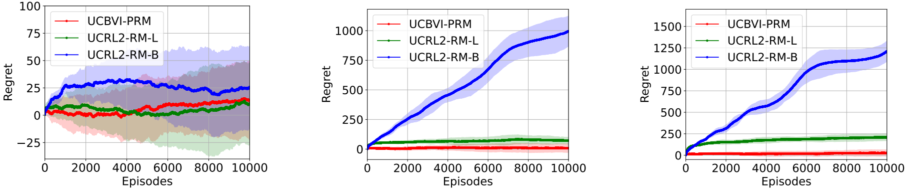
Theorem

With high probability

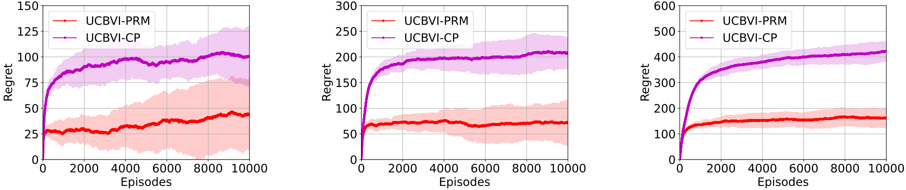
$$Regret(K) \leq \widetilde{O}(\sqrt{OAH^2K}) + \textit{lower order terms}$$

Matches the **lower bound** asymptotically up to a logarithmic factor.

Experiments

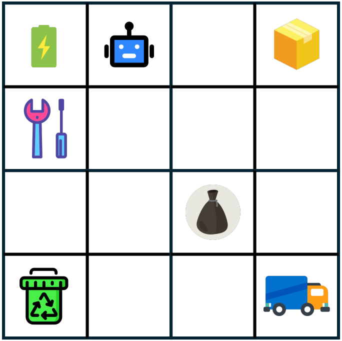


Experimental results in RiverSwim with different length of horizons and number of observations



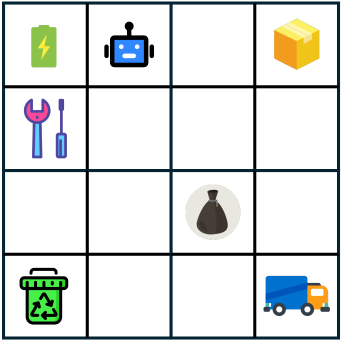
Experimental results in Warehouse with different length of horizons and number of observations

What if we have multiple PRMs?



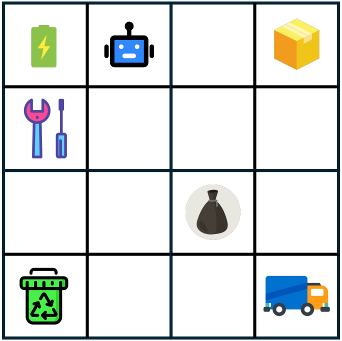
- Task 1: collect  and deliver it to 

What if we have multiple PRMs?



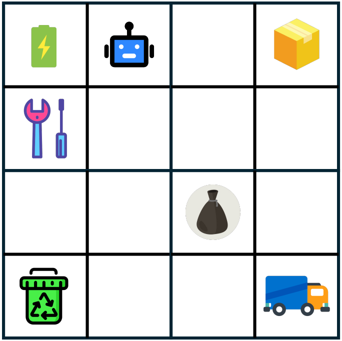
- Task 1: collect  and deliver it to 
- Task 2: collect  and take it to 

What if we have multiple PRMs?



- Task 1: collect  and deliver it to 
- Task 2: collect  and take it to 
- Task 3: go  and go 

What if we have multiple PRMs?



- Task 1: collect  and deliver it to 
- Task 2: collect  and take it to 
- Task 3: go  and go 

......

What if we have multiple PRMs?

As humans, we have numerous requirements for 🤖.

Do we have to run our learning algorithm every time when a new PRM comes up?

Reward-free exploration

- **1. Exploratory Policy Set:**
  - For every (o, a), let $r(o, a) = 1$.
    - Run RL algorithm when $r(o, a) = 1$.
    - Collect policy into Ψ.
- **2. Collect Trajectories:**
  - Sample policy π from Ψ uniformly.
  - Play policy π, collect data and append to D.
- Use data buffer D to estimate transition functions $\hat{p}$.
- Planning under $(\hat{p}, \mathcal{R})$.

Theoretical Guarantee

After $\widetilde{O}(\frac{O^5 A^3 H^2 G^2}{\epsilon^2})$ episodes of exploration and returns an ϵ-optimal policy for any PRMs. G is the largest return for any trajectory.

Can be extendable to any other Non-Markovian rewards if a planner exists!

Summary and Extension

Summary:

- An efficient algorithm tailored for PRMs that matches the lower bound asymptotically.
- Reward-free learning results for non-Markovian rewards.

Extension:

- Multi-agent settings.
- Other reward structures such as submodular rewards.